DEFENSE INFORMATION SYSTEMS AGENCY

EMERGING WEB SERVICES DEVELOPMENT ENVIRONMENT

8 DECEMBER 2003

EXECUTIVE SUMMARY:

Application frameworks are a holistic set of guidelines and specifications that provide platforms, tools, and programming environments for addressing the design, integration, performance, security, and reliability of distributed and multi-tiered applications. The two most prominent application frameworks that exist today are Sun Microsystem's Java 2 Platform, Enterprise Edition (J2EE) and Microsoft's .NET. While .NET was built on the foundation of XML and Web Services, J2EE (which originated before XML 1.0 was a W3C Recommendation and before Web Services came into being) was not built with XML and Web Services at its core but rather has been extended over time to include support for XML and Web Services. J2EE is Java-centric and platform neutral, while .NET is Windows-centric and language-neutral.

The market today is rich with Web Services development tools from varied vendors. Web Services development tools enable software developers to execute tasks such as WSDL document creation, SOAP interface generation, and invocation and testing of Web Services. We discuss products from vendors such as IBM, Systinet, Microsoft, and Sun Microsystems. We also describe a recommended SOAP profile for DISA that is based on the Web Services Interoperability Organization (WS-I) Basic Profile 1.0, and calls for the consideration of incorporating the WS-I Basic Security Profile once available.

There also exist other categories of Web Services products. XML authoring tools (with Web Services features) enable capabilities such as interpretation of WSDL documents, creation of SOAP requests, and a SOAP Debugger. Web Services-based business integration products enable businesses to exchange data across their organizational boundaries, and to potentially integrate processes as well. Web Services monitoring/management products enable the monitoring and management of Web Services as they execute, and the detection and diagnosis of processing exceptions. Finally, Web Services acceleration products are hardware-based products introduced that apply proprietary techniques in order to accelerate the processing of XML-based data and Web Services. We discuss products in these categories from vendors such as Altova, Cape Clear Software, AmberPoint, and DataPower.

While DISA may not require products in all of the categories discussed in this report, we believe that there may be needs in many of the categories. Although we do not make specific product and tool recommendations in this section, we recommend that DISA undertake initiatives to evaluate products in the categories that are discussed.

Table of Contents

1 Introduction

This section addresses Web Services development environments, products, and tools. The intent of this section is not to provide recommendations for any specific products, but rather to demonstrate the breadth of coverage that is available in Web Services products today. In this section we discuss the following topics:

- **Application Frameworks:** The two major application frameworks, Java 2 Platform, Enterprise Edition (J2EE) and Microsoft .NET
- **Web Services Development Tools:** Tools that enable developers to execute tasks such as WSDL document creation, SOAP interface generation, and invocation and testing of Web Services.
- **Miscellaneous Web Services Products:** Diverse products such as XML authoring tools, Web Services-based business integration products, Web Services monitoring/management products, and Web Services acceleration products.

We also provide a recommendation for a SOAP profile for DISA.

2 Application Frameworks

This section describes the two most prominent application frameworks existing today – Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE) and Microsoft's .NET. Application frameworks are a holistic set of guidelines and specifications that provide platforms, tools, and programming environments for addressing the design, integration, performance, security, and reliability of distributed and multi-tiered applications. An application framework includes features such as: presentation services, server-side processing, session management, business logic framework, application data caching, application logic, persistence, transactions, security, and logging services.

2.1 J2EE

2.1.1 Overview

The Java 2 Platform, Enterprise Edition (J2EE) is a series of specifications for developing multi-tier enterprise applications. J2EE technology and its component-based application model provide a simplified approach to developing highly scalable and highly available Web or intranet-based applications. The initiative to develop J2EE was announced by Sun Microsystems in April 1997.

Enterprise applications support core business operations; therefore, any failure of these applications may cause an interruption in business operations. Additionally, the World Wide Web has added a significant level of complexity to enterprise applications, which now must support all business operations, including those conducted with external business partners and customers Some of the definitive characteristics of this class of enterprise applications are: scalability, availability, reliability, security, transactional integrity, and distribution. To support this robust level of service, enterprise applications are often designed using a multi-tier, distributed object architecture, which J2EE provides. A certified J2EE platform provides a consistent, integrated Java runtime environment that guarantees a certain quality of service and ensures application portability and interoperability. The J2EE specifications are defined under the auspices of the Java Community Process as Java Specification Requests, and application servers implement them.

The following is an overview of the J2EE environment – it depicts a multi-tier architecture comprised of a client tier (with which the user interacts directly), a middle tier (in which application logic resides), and an EIS tier (also known as the "data tier") which supports access to existing enterprise information systems and data sources:

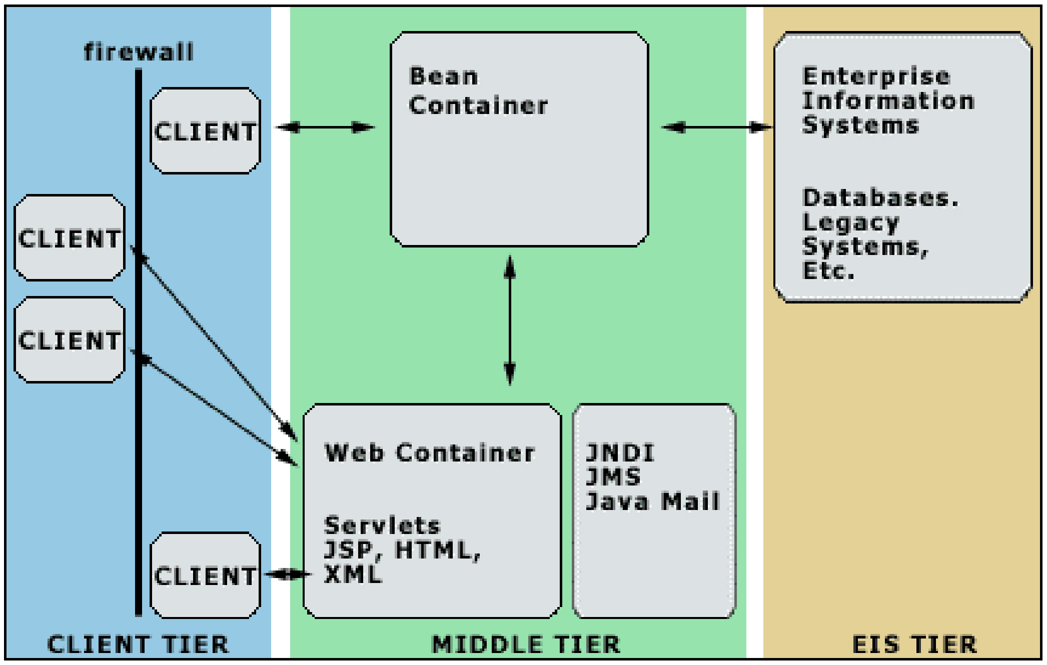

Figure 1

Source: http://www.apexlogic.com/wp_j2ee.html

Many of the components shown above will be described in the sections that follow.

2.1.2 J2EE Components

The J2EE platform is composed of component-based technologies. J2EE components are written in the Java programming language and are compiled in the same way as any program written in the language. The J2EE application model encapsulates the layers of functionality in specific types of components:

- Enterprise JavaBeans (EJB)
- Servlets
- JavaServer Pages (JSP)

Each of these components is discussed in further detail below. Though JavaBeans components are not considered by the J2EE specification to be J2EE components, they are also described.

2.1.2.1 Enterprise Java Beans (EJB)

The Enterprise JavaBeans (EJB) component model simplifies the development of middleware applications by providing automatic support for services such as transactions, security, database connectivity, and more. Business logic is handled by enterprise beans running in the J2EE business (middle) tier.

There are three different types of EJB components: session beans, entity beans, and message-driven beans. **Session beans** represent behaviors associated with client sessions (e.g., a user purchase transaction on an e-commerce site). When the client finishes executing, the session bean and its data are gone. **Entity beans** represent collections of data, such as rows in a relational database, and encapsulate operations on the data they represent. In contrast to session beans, entity beans are persistent, surviving as long as the data they're associated with remains viable. This means that if the client terminates or if the server shuts down, the underlying services ensure that the entity bean data is saved. **Message-driven beans** were introduced as a new component type in the Enterprise JavaBeans 2.0 specification. Message-driven beans combine features of a session bean and a Java Message Service (JMS) message listener, allowing a business component to receive JMS messages asynchronously. The EJB 2.1 specification generalizes the architecture for message-driven beans to support the use of messaging types in addition to JMS.

The EJB 2.1 specification also extends the EJB 2.0 specification with support for Web Services. This includes the ability for a stateless session bean to implement a Web Service endpoint, and for all enterprise beans to utilize external Web Services. The Web Service endpoint that is implemented by a stateless session bean is described by a WSDL document and accessed by Java clients through the JAX-RPC client APIs, which utilize the SOAP 1.1 protocol over an HTTP transport. Only a stateless session bean may provide a Web Service endpoint interface.

2.1.2.2 Servlets

A servlet is a small Java technology-based program that runs on a server and generates dynamic content. The term was "servlet" coined in the context of the Java applet, a small program that is sent as a separate file along with a Web (HTML) page and which is intended to run in a client. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Servlets are managed by containers, which are Web server extensions that provide servlet functionality, and which are sometimes called "servlet engines". Servlets interact with Web clients via a request/response paradigm implemented by the servlet container – they process Web requests on the server, pass them into the back-end enterprise application systems, and dynamically render the results as HTML or XML client interfaces.

2.1.2.3 JavaServer Pages (JSP)

JavaServer Pages (JSP) enables the creation of dynamic Web pages that can incorporate Java programming language statements. The main features of JSP technology are:

- A language for developing JSP pages
- An expression language for accessing server-side objects
- Mechanisms for defining extensions to the JSP language

JSP controls the content or appearance of Web pages through the use of servlets, which modify the Web page before it is sent to the user who requested it. JSP is comparable to Microsoft's Active Server Page (ASP) technology; however, whereas a Java Server Page calls a Java program that is executed by the Web server, an Active Server Page contains a script that is interpreted by a script interpreter (such as VBScript or JScript) before the page is sent to the user. JSP scripting elements allow the use of Java programming language statements in JSP pages. Additionally, the JavaServer Pages Standard Tag Library (JSTL), which encapsulates core functionality common to many JSP applications, can be used within JSP Pages as an expression language.

2.1.2.4 JavaBeans

A JavaBean is a reusable software component (such as a slider box) that can be visually manipulated in builder tools. Text tools through programmatic interfaces can also manually manipulate Beans. A JavaBean is similar in concept to an ActiveX control, which is part of the ActiveX set of COM (Component Object Model)-based technologies originally developed by Microsoft. The principal difference between ActiveX controls and JavaBeans are that ActiveX controls can be developed in any programming language but executed only on a Windows platform, whereas JavaBeans can be developed only in Java, but can run on any platform.

2.1.3 J2EE and Web Services

Web Services become an integral part of the J2EE platform with J2EE 1.4. Prior to that time, various vendors had created products that integrated with J2EE application servers to provide support for Web Services; however, these implementations were not governed by a set of specifications, and J2EE Web Services applications were thus bound to the J2EE product upon which they were deployed. In April 2002, a "Web Services for J2EE" specification was created for implementation on top of a J2EE 1.3 platform. The section below provides more information about this specification. It is followed by a description of Web Services capabilities in J2EE 1.4.

2.1.3.1 Web Services for J2EE Specification

The "Web Services for J2EE" specification defined standard APIs to enable programmers to develop Web Services applications in the Java programming language. These APIs were grouped together as the "Java APIs for XML". The Web Services for J2EE specification used the Java API for XML-Based RPC (JAX-RPC) specification as a base technology. Some of the objectives of the Web Services for J2EE specification were:

- Leveraging technologies defined by the existing specifications
- Defining minimum new concepts in the specification
- Supporting a simple model for development of a Web Service and its deployment in a J2EE application server
- Defining functions that application server vendors need to support to comply with the Web Services for J2EE specification
- Defining platform roles (e.g. developer, assembler, deployer) and their mapping to existing J2EE platform roles

The Web Services for J2EE specification provided new definitions, or additional requirements for existing definitions, in a number of areas, such as:

- Defining a Web Service

- Discovering and accessing a Web Service
- Implementation details of a Web Service and a Web Services client
- Packaging and deployment descriptor format
- Deployment process details and requirements
- State and lifecycle management of a Web Service, and the responsibilities of the container (either servlet or EJB) where the Web Service is deployed
- Runtime behavior of a Web Service

The figure below illustrates how the Web Services for J2EE specification defines a Web Service:

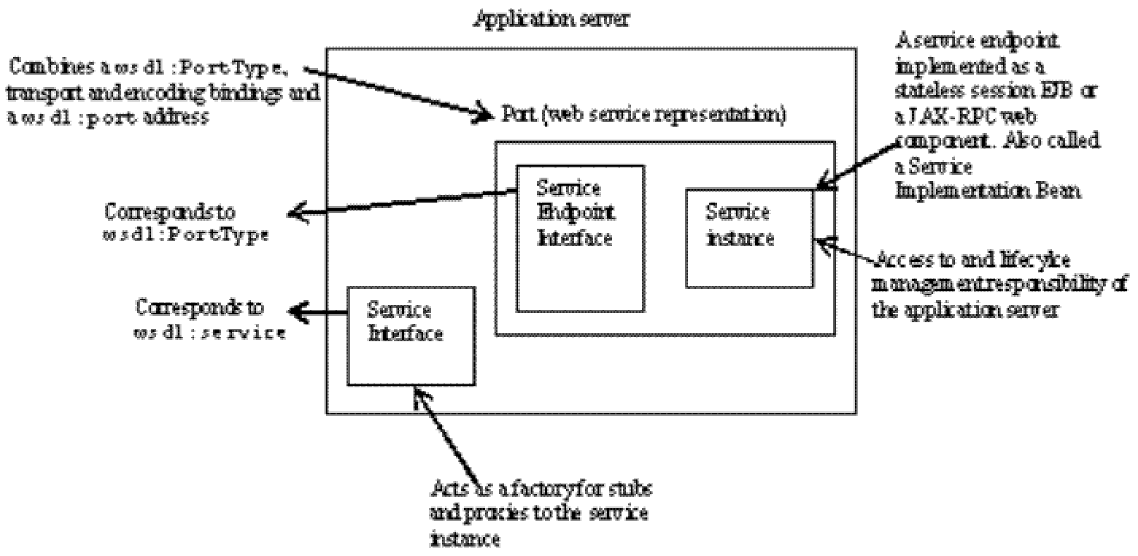
Application server
Combines a wsdl:PortType, transport and encoding bindings and a wsdl:port address
Port (web service representation)
A service endpoint implemented as a stateless session EJB or a JAX-RPC web component. Also called a Service Implementation Bean
Corresponds to wsdl:PortType
Service Endpoint Interface
Service instance
Access to and lifecycle management responsibility of the application server
Corresponds to wsdl:service
Service Interface
Acts as a factory for stubs and proxies to the service instance

Figure 2

Source: http://www.webservices.org

As shown in the above figure, the Web Services for J2EE specification defines a Web Service in terms of a Port component. A Port component is a representation of a Web Service, and consists of the service endpoint interface, as well as service instance that implements the methods of the service endpoint interface. The service endpoint interface (also known as a service implementation bean) contains the business logic of the Web Service, and it is defined by the JAX-RPC specification. A service endpoint interface can be implemented either as a Java class that runs as part of a Web component deployed in a servlet container, or a stateless session bean that will be deployed in an EJB container.

2.1.3.2 J2EE 1.4

The J2EE 1.4 platform, approved by the Java Community Process (JCP) on November 18, 2003, introduces full, standard integration of Web Services in J2EE for the first time. The J2EE 1.4 platform employs various XML APIs and tools for the design, development, test, and deployment of Web Services and clients that fully interoperate with other Web Services and clients running on Java-based or non-Java-based platforms. With J2EE 1.4, there is less of a need to convert data between different formats as a Web-services request comes into a J2EE application server and is processed by a Java component. The XML API implementations in J2EE 1.4 alleviate the need for low-level programming because they will translate the application data to and from an XML-based data stream that is sent over the standardized XML-based transport protocols. The transported data can itself be plain text, XML data, or any kind of binary data such as audio, video, maps, program files, CAD documents or the like. J2EE 1.4 will also support the Web Services Interoperability (WS-I) Basic Profile specification.

The various Java APIs are each described below. Although not an XML API, we also describe the Java Messaging Service API for comparison purposes.

2.1.3.2.1 Java API for XML-Based RPC (JAX-RPC)

Java API for XML-Based RPC (JAX-RPC) is an API for building Web Services and clients that use remote procedure calls (RPC) and XML. It defines a SOAP-based RPC mechanism that enables client programs to make XML-based remote procedure calls (RPCs) over the Web, as well as the usage and manipulation of WSDL documents through a WSDL-to-Java mapping. JAX-RPC hides the complexity of SOAP messages from the application developer, and uses servlets behind-the-scenes to implement the Web Services.

JAX-RPC defines the following:

- A client API to enable Java-based clients to access Web Services, as well as a client-side runtime for SOAP-based invocation of Web Services. The client-side runtime takes care of a number of issues, such as mapping between XML and Java types, creation of a SOAP message, and sending it over HTTP.
- WSDL-to-Java mapping for generation of client-side artifacts to enable access to Web Services. Used by the Java clients to invoke methods on a Web Service.
- Java-to-WSDL mapping for exposing existing Java-based service endpoints as Web Services. This allows an existing object to be mapped to a WSDL description, which could then be used by potential clients to invoke methods on the object.
- Support for SOAP 1.1 binding and HTTP 1.1 as transport protocol.
- Support for SOAP message processing on both client and server sides.
- A servlet container-based JAX-RPC service endpoint model on the service side, where the Web Service is to be deployed in a servlet container. The service is not implemented as a servlet; rather, it is associated at deployment time with a servlet provided by the JAX-RPC runtime. The servlet would, for example, handle transport-specific processing of the incoming request and dispatching the request to the service endpoint.

The following figure illustrates the above concepts:

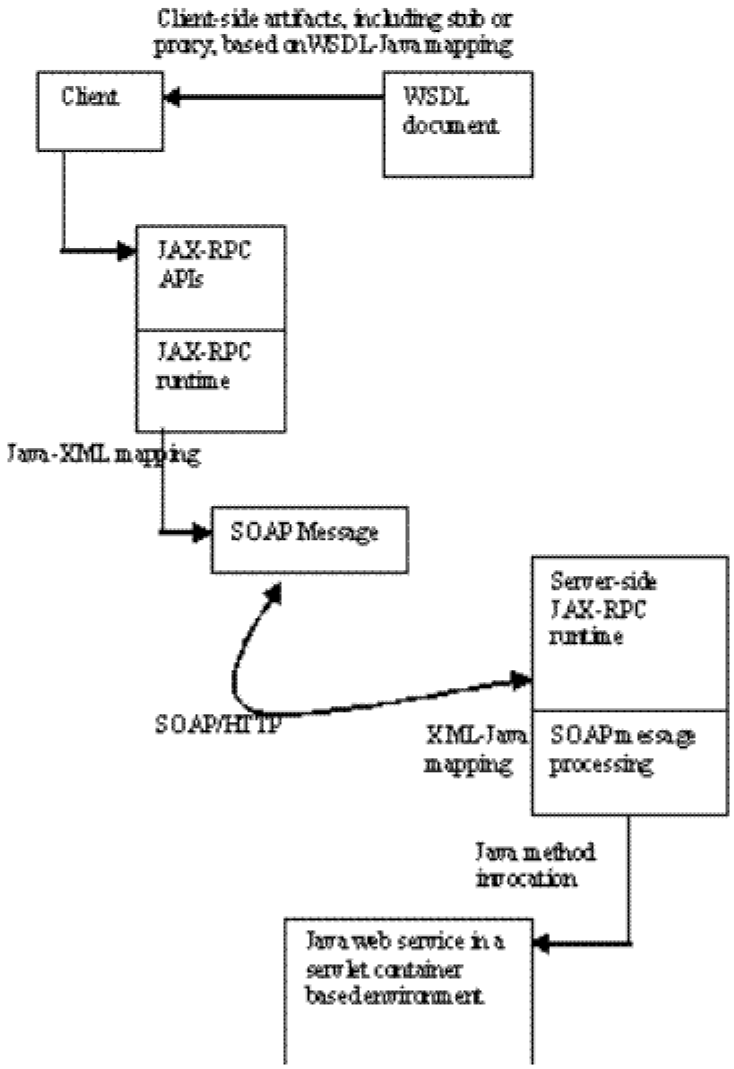
Client-side artifacts, including stub or proxy, based on WSDL-Java mapping
Client
WSDL document
JAX-RPC APIs
JAX-RPC runtime
Java-XML mapping
SOAP Message
Server-side JAX-RPC runtime
SOAP/HTTP
XML-Java mapping
SOAP message processing
Java method invocation
Java web service in a servlet container based environment

Figure 3

Source: http://www.webservices.org

JAX-RPC does not define an EJB container-based service endpoint model, or how Web Services could be developed using the EJB component model; rather, this is left for the Web Services for J2EE specification, which essentially picks up from where JAX-RPC leaves off. With JAX-RPC and a WSDL document, one can easily interoperate with clients and services running on Java-based or non-Java-based platforms such as .NET. For example, based on a WSDL document, a Visual Basic .NET client can be configured to use a Web Service implemented in Java technology, or a Web Service can be configured to recognize a Visual Basic .NET client.

The following lists the various elements of a WSDL document and how they map to Java artifacts as defined by the JAX-RPC specification:

- **wsdl:operation:** Names an operation and lists the expected inputs and outputs. It maps to a Java method.
- **wsdl:portType:** Contains a set of one or more operations. Corresponds to a JAX-RPC service endpoint interface; the Web Service implementation then implements the methods of the service endpoint interface.
- **wsdl:port:** Specifies an address for a service port (or endpoint) for a given protocol binding, e.g. SOAP.
- **wsdl:service:** Groups a set of service endpoints, or ports. Maps to a JAX-RPC service interface, which acts as a factory for a stub or proxy to the service endpoint or port.

2.1.3.2.2 Java API for XML Messaging (JAXM)

The Java API for XML Messaging (JAXM) provides support for sending and receiving document-oriented XML messages via SOAP with Attachments. JAXM 1.0 was the first Java API to support SOAP; it therefore pre-dates JAX-RPC. However, JAXM is document-oriented while JAX-RPC is RPC-oriented.

The JAXM API comprises the following APIs: the SAAJ 1.1 API and JAXM 1.1 API. The SAAJ 1.1 API (SOAP with Attachments API for Java) defines interfaces for constructing and populating a SOAP message, while the JAXM 1.1 API defines interfaces for sending request/response messages and depends on the SAAJ 1.1 API for constructing and populating the SOAP messages. Both JAXM and JAX-RPC rely on SAAJ 1.1.

The following figure depicts a JAXM API between a Java application and multiple vendors using JAXM. Each vendor utilizes their own JAXM messaging provider that manages the routing of messages for their JAXM clients:

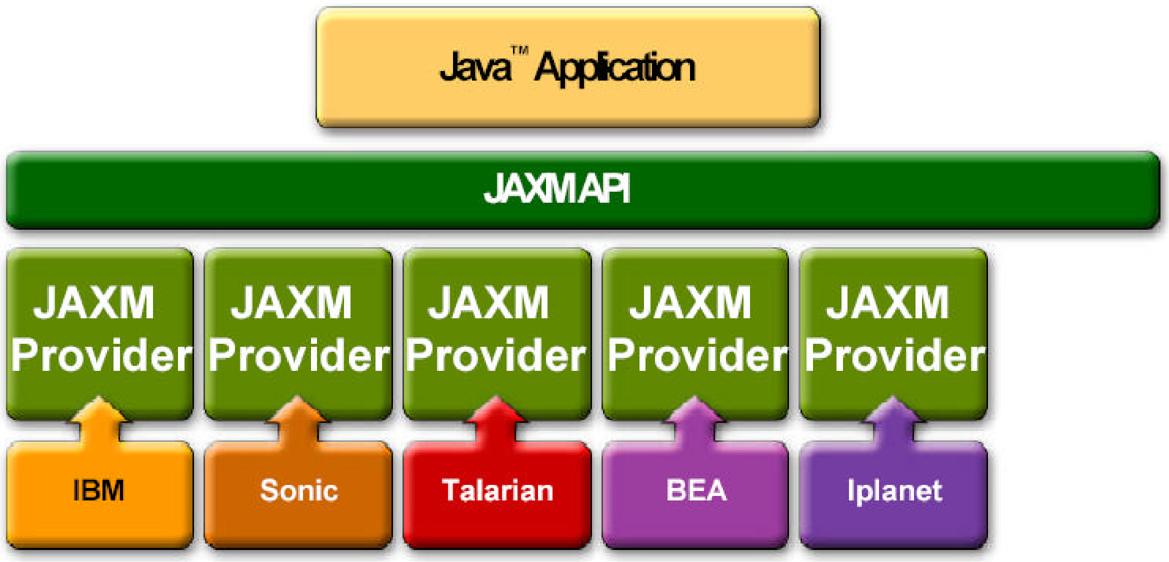

Figure 4

Source: http://developer.java.sun.com

2.1.3.2.3 Java Architecture for XML Binding (JAXB):

The Java Architecture for XML Binding (JAXB) provides an API and tools that automate the mapping between XML documents and Java objects – thus it provides a bridge between XML and Java technology. JAXB makes XML easy to use by compiling an XML schema into one or more Java classes. Thus, JAXB enables the creation of Java objects at the same conceptual level as the XML data. Representing data in this manner enables manipulation of the data in the same manner in which Java objects would be manipulated, thereby making it easier to create applications to process XML data. Additionally, after working with the data, the Java objects can be written the to new, valid XML document.

The following figures depicts the JAXB data binding process:

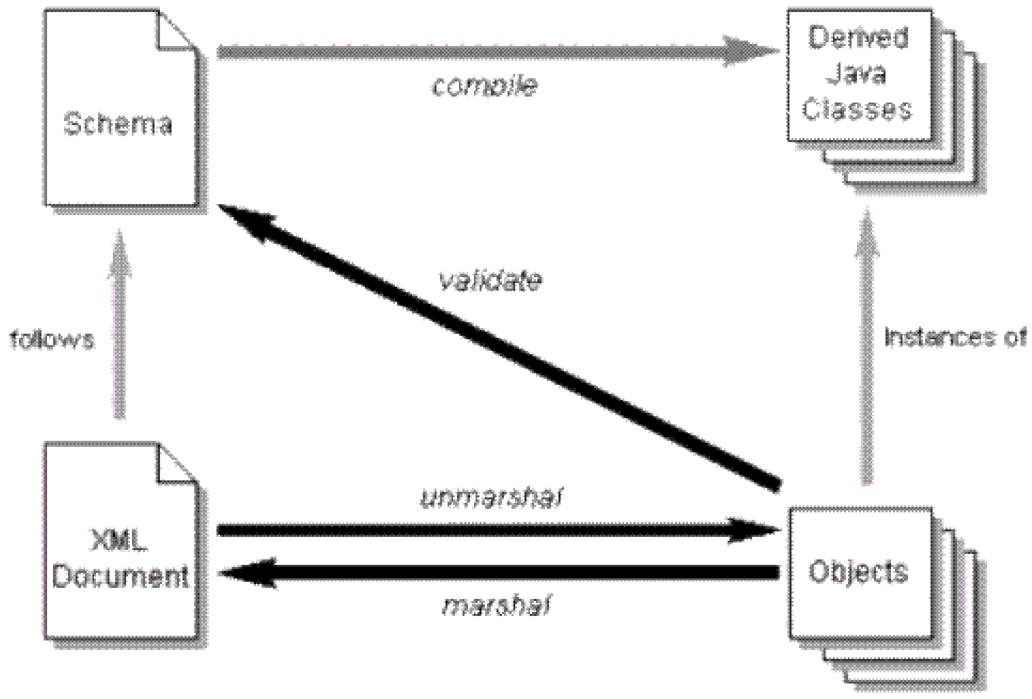

Figure 5

Source: http://java.sun.com

2.1.3.2.4 Java API for XML Processing (JAXP):

The Java API for XML Processing (JAXP) enables the processing of XML data using applications written in the Java programming language. JAXP supports the processing of XML documents using various techniques such as W3C Document Object Model (DOM) and Simple API for XML Parsing (SAX) for parsing XML documents, and W3C XML Style sheet Language Transformation (XSLT) for conversion of data to other XML documents or to other formats such as HTML. JAXP also provides the option of data validation.

Although they may appear to overlap in capabilities, JAXB and JAXP actually serve very different purposes. JAXP is used for transformation of data from one format to another using the same set of APIs, while JAXB is used for building object representations of XML data.

2.1.3.2.5 Java API for XML Registries (JAXR):

The Java API for XML Registries (JAXR) enables the use of a single API to access a variety of XML registries, for operations such as the discovery and retrieval of Web Services descriptions (e.g. WSDL documents). A unified JAXR information model describes content and metadata within XML registries, and gives developers the ability to write registry client programs that are portable across different target registries. JAXR also enables value-added capabilities beyond those of the underlying registries. The current version of the JAXR specification includes detailed bindings between the JAXR information model and both the ebXML Registry and the UDDI version 2 specifications.

The following figure depicts the JAXR architecture:

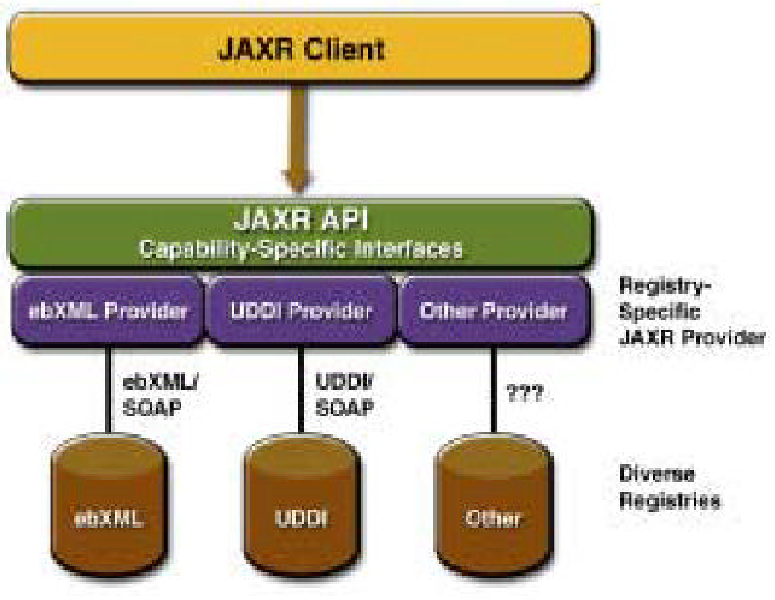

Figure 10–1 JAXR Architecture

Figure 6

Source: http://java.sun.com

2.1.3.2.6 Java Messaging Service (JMS):

The Java Message Service (JMS) API defines a standard mechanism for J2EE application components to send and receive messages in a loosely coupled, asynchronous, and reliable manner. The JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations. The JMS Specification was first published in August

1998. The latest version of the JMS Specification is Version 1.1, which was released in April 2002.

A JMS application is composed of the following parts:

- **JMS Provider:** A messaging system that implements the JMS interfaces and provides administrative and control features.
- **JMS Clients:** The Java programs or components that produce and consume messages. Any J2EE component can act as a JMS client.
- **Messages:** The objects that communicate information between JMS clients.
- **Administered objects:** Preconfigured JMS objects created by an administrator for the use of clients.

Administrative tools enable the binding of administered objects into a Java Naming and Directory Interface (JNDI) API namespace. A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

The figure below illustrates the way these various parts interact (NOTE: The “CF” and “D” references within the JNDI Namespace stand for “connection factories” and “destinations”, which are the two types of administered objects):

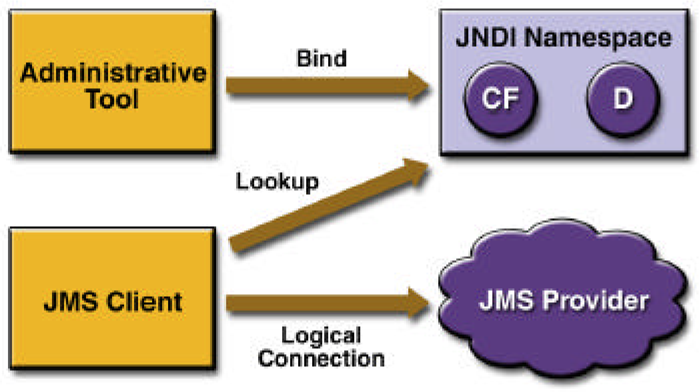

Figure 20–2 JMS API Architecture

Figure 7

Source: http://java.sun.com

Prior to the JMS API, most messaging products supported either the “point-to-point” or “publish/subscribe” approach to messaging. A point-to-point product or application is built around the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.

The following figure depicts point-to-point messaging:

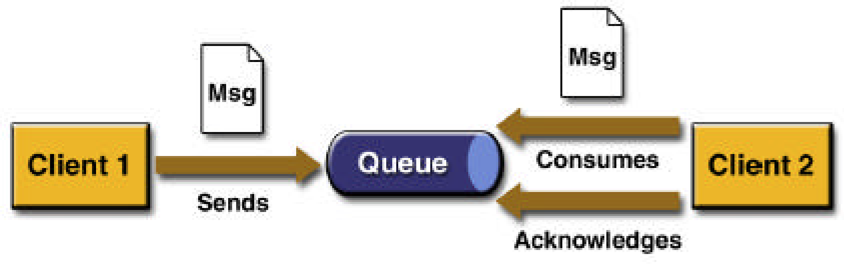

Figure 20–3 Point-to-Point Messaging

Figure 8

Source: http://java.sun.com

In a publish/subscribe product or application, clients address messages to a topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy, and the system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.

The following figure depicts publish/subscribe messaging:

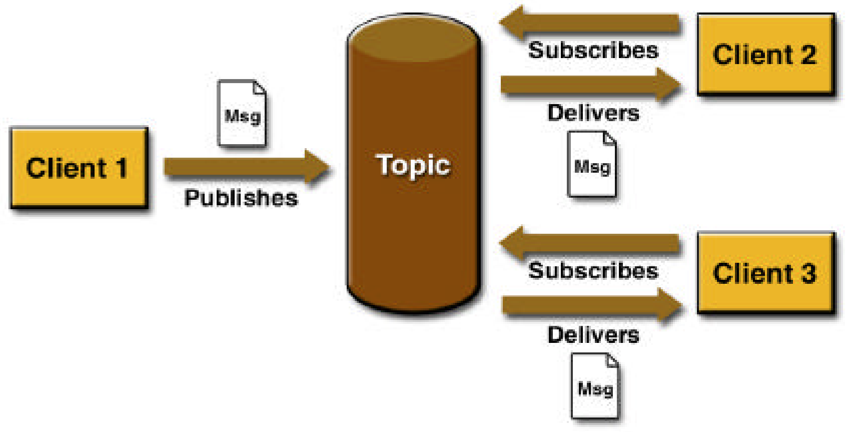

Figure 20–4 Publish/Subscribe Messaging

Figure 9

Source: http://java.sun.com

The JMS specification provides a separate domain for each approach and defines compliance for each domain, while also providing common interfaces that enable the use of the JMS API in a way that is not specific to either domain. A “standalone” JMS provider may implement one or both domains, while a J2EE provider must implement both domains.

While JAXM and JMS both support reliable, asynchronous, secure message delivery and routing, a JAXM client is interoperable over any SOAP 1.1-compatible client while a JMS client is interoperable only with another JMS client over the same messaging system. However, JMS implementations typically work between JMS provides across the Web via SOAP-based JMS-to-JMS communication. Both JAXM and JAX-RPC can be used in conjunction with JMS for transport of XML messages.

2.1.4 J2EE Application Servers

Application servers are middle-tier software that runs between Web-based, thin browser clients and backend databases and applications. This section describes several J2EE-based application servers that are available today.

2.1.4.1 BEA Web Logic

BEA WebLogic Server features a fully J2EE-compliant services oriented architecture and support for rich tool sets. Its comprehensive capabilities support an integrated infrastructure that can connect legacy systems, as well as the latest Web Services. BEA WebLogic Server is a cornerstone of the BEA WebLogic Enterprise Platform, which is shown in the following figure:

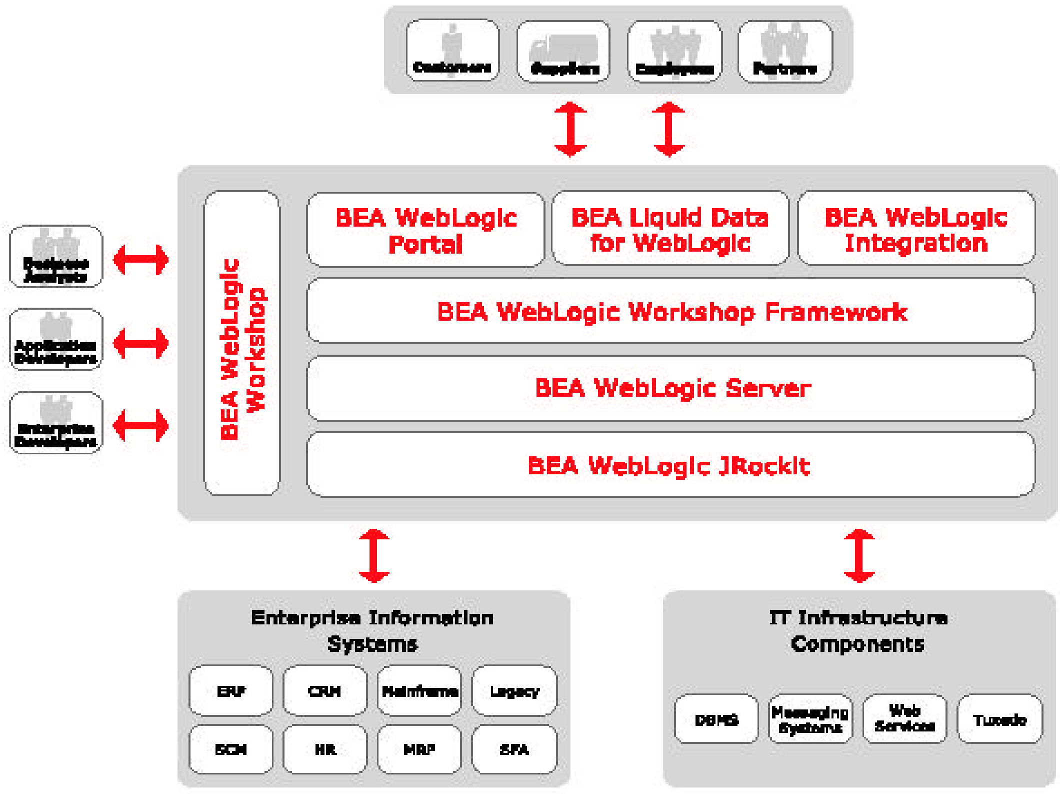

Figure 10

Source: http://www.bea.com

The BEA WebLogic Enterprise Platform is an integrated suite of application infrastructure built on open standards that is designed to allow data and services to be shared across multiple applications and business functions. It includes the following components:

- **BEA WebLogic Server:** A J2EE-compliant application server.
- **BEA WebLogic Workshop:** A unified graphical development environment and run-time framework that enables application developers to rapidly create, test, and deploy enterprise-class Web Service applications on the BEA WebLogic Enterprise Platform.
- **BEA WebLogic Integration:** A single solution delivering application server, application integration, business process management, and B2B integration functionality for the enterprise.
- **BEA WebLogic Portal:** A fully integrated enterprise platform that includes a portal framework with portal foundation services, personalization and interaction management, integration services, and intelligent administration.
- **BEA Liquid Data for WebLogic:** A virtual data access and aggregation product for information visibility that allows a real-time unified view of disparate enterprise data.
- **BEA Tuxedo:** A transaction processing platform built on a service oriented architecture.

- BEA WebLogic JRockit: A high-performance Java Virtual Machine optimized to run on Intel 32 and 64 platforms.

BEA WebLogic Server 8.1 is fully compliant with J2EE 1.3, and includes following features:

- Support for WS-Security
- A "reliable SOAP" feature that provides functionality usually found in reliable messaging frameworks, such guaranteed delivery, "exactly-once" delivery, ordered conversation, etc.
- A built-in messaging infrastructure that implements the JMS specification

2.1.4.2 IBM WebSphere

The IBM WebSphere Application Server is a J2EE-based and Web Services technology-based application platform that is part of a group of products called the WebSphere family of products. It is fully J2EE 1.3 compatible. The WebSphere family of products is comprised of the following categories:

- **Foundation and tools:** Products that represent the basic functional infrastructure for other products. Includes WebSphere MQ and WebSphere Studio in addition to WebSphere Application Server.
- **Reach and user experience:** Products that help the applications extend their range and reach new customers. Includes WebSphere Portal, WebSphere Everyplace, and WebSphere Commerce.
- **Business integration:** Products that help customers to interconnect their islands of information and make full use of a message-based architecture. Includes WebSphere Business Integration and WebSphere MQ Integration Broker.

Each of the above products is briefly described below.

- **WebSphere MQ:** Integrates business application and processes across an extended enterprise through JMS support.
- **WebSphere Studio:** A set of development tools for enterprise Web-based applications.
- **WebSphere Portal:** Enables the development of business-to-employee (B2E), business-to-business (B2B) and business-to-consumer (B2C) portals.
- **WebSphere Everyplace:** Enables users to interchange any information over any network using any device, including Web-enabled cellular phones and other mobile devices.
- **WebSphere Commerce:** IBM's online trading solution that enables creation of online stores.
- **WebSphere Business Integration:** Provides end-to-end integration through component software, connectors, and adapters. Also contains the WebSphere MQ

Integration Broker, which provides a scalable, event-driven application integration model.

WebSphere Application Server also contains support for the JDK 1.4 client container, as a first step toward J2EE 1.4 compliance. It also supports the WS-I Basic Profile specification.

2.1.4.3 Sun ONE Application Server

The Sun ONE Application Server (formerly iPlanet Application Server) is part of the Sun ONE (Open Net Environment) architecture. It features a fully J2EE-compliant services oriented architecture and support for rich tool sets, and includes JavaServer Pages support for presentation logic, Java Servlets for server side logic, Enterprise Java Beans (EJBs) for core business logic, and JDBC for access to relational databases.
Sun ONE is Sun's standards-based software vision, architecture, platform, and expertise for building and deploying Services on Demand. The following figure depicts the Sun ONE architecture, with a centered emphasis on Web Services:

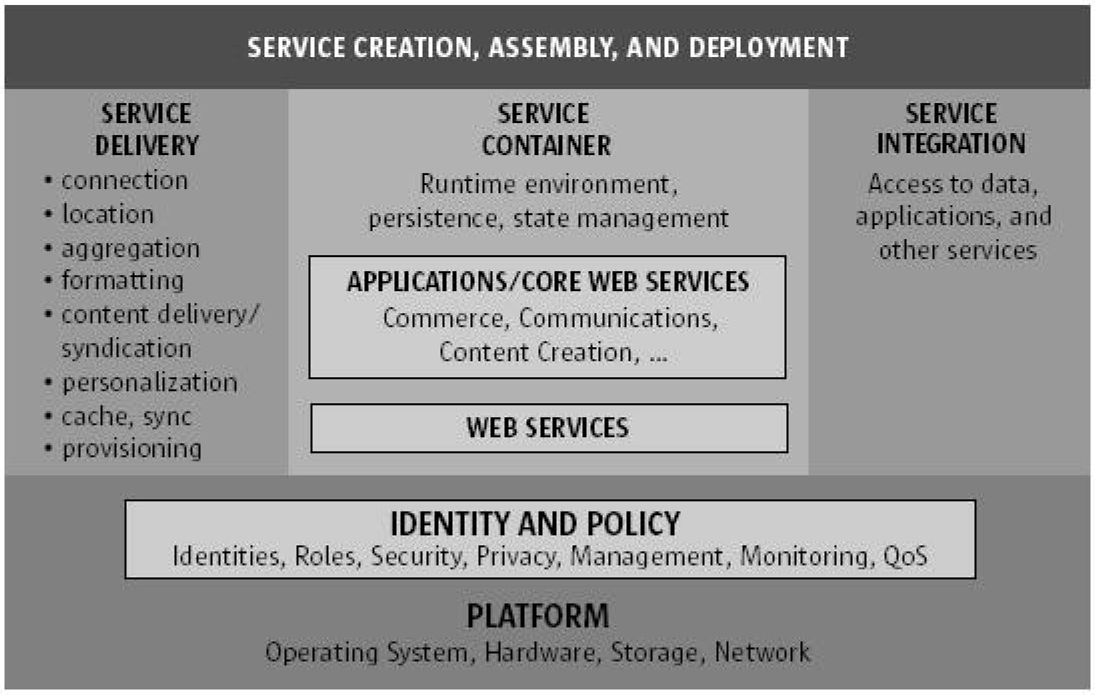
SERVICE CREATION, ASSEMBLY, AND DEPLOYMENT
SERVICE DELIVERY
• connection
• location
• aggregation
• formatting
• content delivery/ syndication
• personalization
• cache, sync
• provisioning
SERVICE CONTAINER
Runtime environment, persistence, state management
APPLICATIONS/CORE WEB SERVICES
Commerce, Communications, Content Creation, ...
WEB SERVICES
SERVICE INTEGRATION
Access to data, applications, and other services
IDENTITY AND POLICY
Identities, Roles, Security, Privacy, Management, Monitoring, QoS
PLATFORM
Operating System, Hardware, Storage, Network

Figure 11

Source: http://www.sun.com

The Sun Web Services Integrated Development Environment (IDE) product is Sun One Studio, formerly Forte. Sun One Studio's foundation is based on NetBeans, which is an open source tools platform.

2.1.4.4 Oracle 9*i* Application Server

Oracle9*i*AS provides a complete Java 2 Enterprise Edition (J2EE) environment written entirely in Java that executes on the Java virtual machine (JVM) of the standard Java Development Kit (JDK). Oracle9*i*AS Containers for J2EE (OC4J) is J2EE 1.3 certified and provides all the containers, APIs, and services that J2EE 1.3 specifies. OC4J is based on technology licensed from Ironflare Corporation, which develops the Orion server - one of the leading J2EE containers.

2.1.4.5 JBoss

JBoss is a free, open-source, Java-based application server. It began in 1999 as an open source EJB container. JBoss is supported by the JBoss Group, a worldwide organization founded by the core developers of JBoss that is dedicated to the professional servicing of JBoss technology.

JBoss includes a JMS implementation called JQossMQ. The JBossGroup announced in November 2003 that it had begun J2EE certification of JBoss by testing JBoss' J2EE 1.4 compatibility using Sun's J2EE 1.4 Compatibility Test Suite (CTS).

2.2 References

"Web Services for J2EE":
http://www.webservices.org/index.php/article/articleview/500/

Java 2 Platform, Enterprise Edition (J2EE):
http://java.sun.com/j2ee/

The Java APIs and Architectures for XML:
http://java.sun.com/xml/webservices.pdf

"J2EE vs. .Net high level overview":
http://www.webservices.org/index.php/article/articleview/258/

JSR 153: Enterprise JavaBeans 2.1:
http://jcp.org/en/jsr/detail?id=153

"J2EE to Support WS-I's Basic Profile":
http://www.eweek.com/article2/0,3959,862575,00.asp

2.3 .NET

Microsoft .NET is a set of software technologies built on the foundation of XML and Web Services. .NET enables the creation and use of XML-based applications, processes, and Web sites as services that share and combine information and functionality with each other by design, on any platform or smart device, to provide tailored solutions for organizations and individuals.

.NET consists of the following technologies:

- **.NET Framework 1.1:** Used for building and running all kinds of software, including Web-based applications, smart client applications, and XML Web Services.

- **Developer tools:** Tools such as Microsoft Visual Studio .NET 2003 which provides an integrated development environment (IDE) for maximizing developer productivity with the .NET Framework.
- **Servers:** A set of servers that that integrate, run, operate, and manage Web Services and Web-based applications, including includes Microsoft Windows® Server 2003, Microsoft SQL Server™, and Microsoft BizTalk® Server.
- **Client software:** Products such as Windows XP, Windows CE, and Microsoft Office XP.

Version 1.0 of the .NET Framework was released in January 2002.

The following figure depicts the .NET Framework:

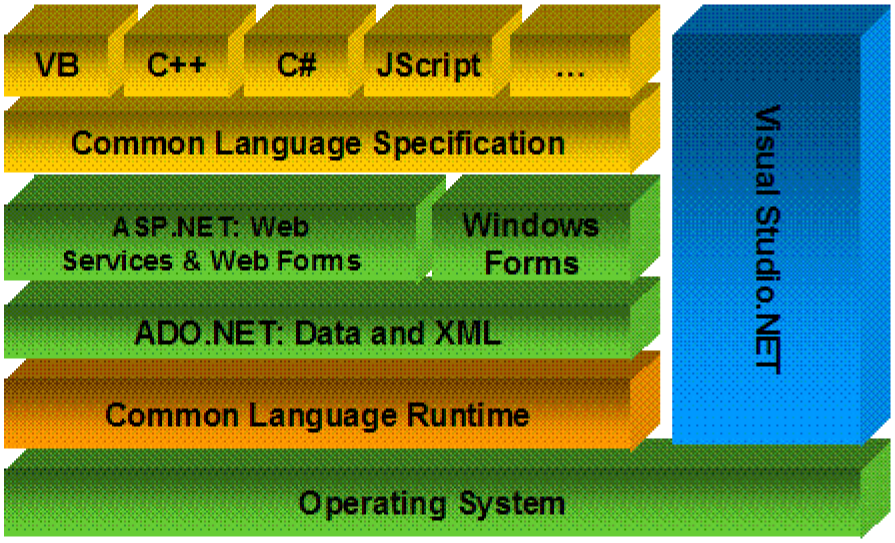

Figure 12

Source: Microsoft

The .NET Framework consists of two main parts: the common language runtime (CLR) and a unified set of class libraries, including ASP.NET for Web applications and Web Services, Windows Forms for smart client applications, and ADO.NET for loosely coupled data access. The .NET Framework will be described in further detail in the sections that follow.

2.3.1 Common Language Runtime (CLR)

The common language runtime allows individual components of specific applications to seamlessly communicate through a standard set of metadata and common execution environment that integrates multiple programming languages and allows objects created in one language to be read with equal weight by code written in a different language. For example, a scheduling function written in COBOL could be used with a human resources application that was written in Microsoft Visual Basic .NET. Additionally, various features of the common language runtime that allow it to manage memory, security, and language integration dramatically reduce the amount of code a developer must write. The multiple-language capability of the .NET Framework through the CLR enables developers to use the programming language that is most appropriate for a given task and to combine languages within a single application.

2.3.2 Class Libraries

The .NET class libraries provide a common, consistent development interface across all languages supported by the .NET Framework. The following class libraries are provided:

- **Base Classes:** Provide standard functionality such as input/output, string manipulation, security management, network communications, thread management, text management, and user interface design features.
- **ADO.NET Classes:** Enable developers to interact with data accessed in the form of XML through the OLE DB, ODBC, Oracle, and SQL Server interfaces.
- **XML Classes:** Enable XML manipulation, searching, and translations.
- **ASP.NET Classes:** Support the development of Web-based applications and Web Services.
- **Windows Forms Classes:** Support the development of desktop-based smart client applications.

Several of these classes are described in further detail below.

2.3.2.1 ADO.NET

ADO.NET is the .NET version of Microsoft's ActiveX Data Objects (ADO) object model. ADO itself (along with ODBC and OLE DB) is part of Microsoft Data Access Components (MDAC), which is a set of technologies that enables universal data access. The following figure illustrates the ADO programming model:

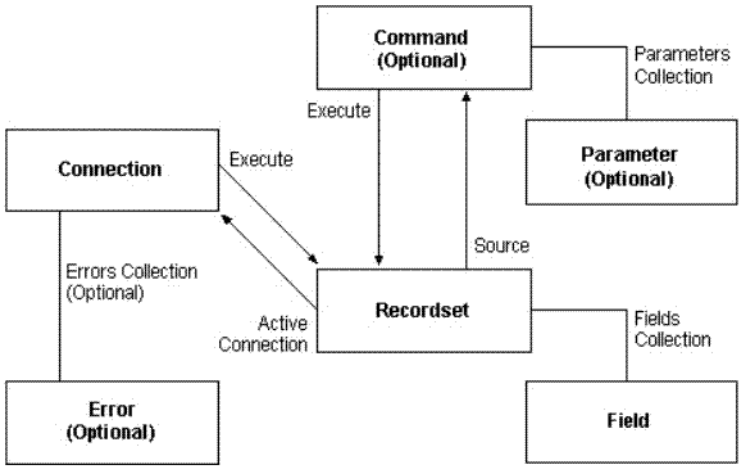

Figure 13

Source: Microsoft

In the above figure, a **Connection object** represents a connection to a data source, such as a relational database. A **Command object** issues commands that can be processed by data source and that may result in a **Recordset** object being returned with command results. Such commands may be simple SQL statements, or stored procedures calls. A **Parameter object** represents a parameter of a command, such as a timeout value for a connection. A **Field object** represents a column (field) in a recordset that can be used to obtain values, modify values, and learn about column metadata. Finally, an **Error object** represents an error returned from a data source.

Unlike ADO, ADO.NET was designed from the ground up to integrate with XML. ADO.NET provides developers an option of working with a platform-neutral, XML-based cache of requested data, instead of directly manipulating a database. ADO.NET persists and loads data and its relational structure as XML, and it transfers data between .NET clients in XML format. While ADO was focused on the concept of a Recordset, ADO.NET introduces a new **DataSet object** that serves as a common, in-memory representation for working with all types of data. A DataSet is, by design, disconnected at all times from outside resources, making it ideal for packaging, exchanging, caching, persisting, and loading data. While a Recordset is database-oriented, a DataSet object is completely independent of a data source (i.e. the data source may be a relational database, an XML document, an application, etc.) and it can behave in many different ways depending on how it is generated and what data it is accessing.

The following figure illustrates the components of the ADO.NET architecture:

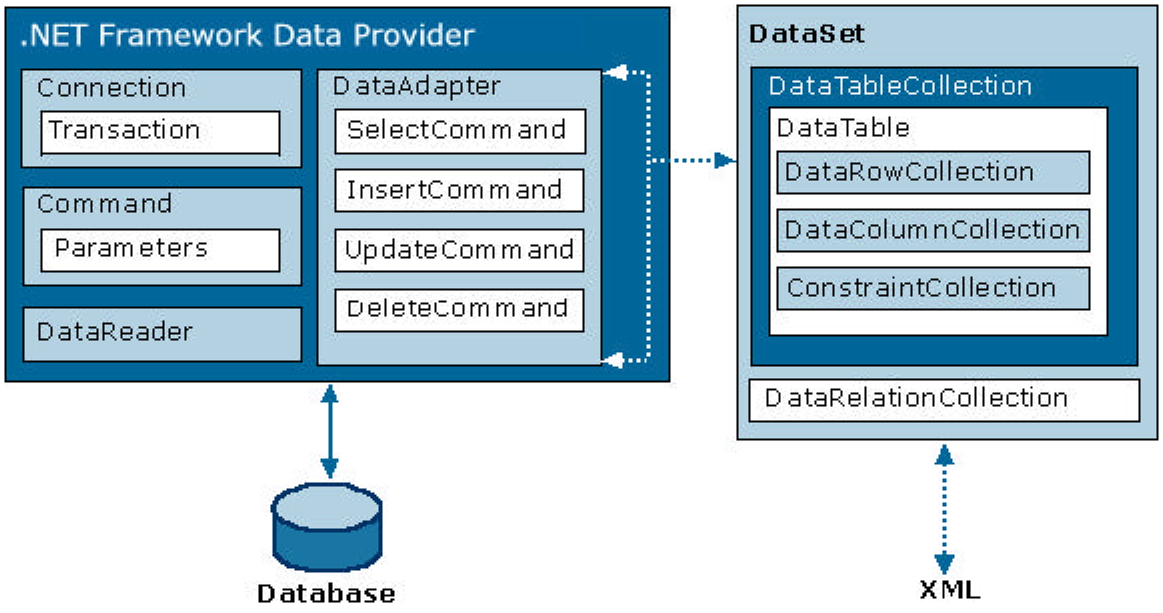

Figure 14

Source: Microsoft

The left side of the above figure depicts interactions with a relational database whose results are ultimately populated into a DataSet. The **DataReader** object (bottom left) is used to retrieve a read-only, forward-only stream of data from a database; results are returned as a query executes, rather than waiting for the completion of the query. The **DataAdapter** constructor provides a bridge between the DataSet object and the data source, and it uses various **Command** objects to execute SQL commands at the data source to load the DataSet with data and to reconcile changes made to the DataSet back to the data source. The **DataTableCollection** class (right) represents the collection of tables for the DataSet, while the **DataTable** class represents a single table of in-memory data.

2.3.2.2 ASP.NET

ASP.NET is the .NET version of Microsoft's Active Server Pages (ASP) technology. ASP is a server-side scripting environment that is used to create dynamic, interactive Web server applications. ASP pages are essentially HTML pages with scripts embedded in them. ASP natively supports both Visual Basic, Scripting Edition (VBScript) and JScript as its scripting languages.

ASP.NET is a programming framework built on the common language runtime that can be used on a server to build powerful Web applications. It includes Web Services to link applications, services and devices using HTTP, HTML, XML and SOAP. Unlike ASP, ASP.NET is language-independent, so developers can choose the language that best applies to their application or partition their application across many languages. Any class can be converted into an XML Web Service with just a few lines of code, and can be called by any SOAP client. Web Services are invoked from an ASP.NET Web page through a Proxy class (created using Visual Studio .NET) that allows a Web Service's methods to be called as if the Web Service resided locally. The Proxy class handles the actual HTTP request as well as the marshalling of the input and output parameters.

2.3.2.3 Windows Forms

Windows Forms is a programming model for writing GUI applications for .NET using the common language runtime and a standalone "thick client". Mostly applications written using Windows Forms require less coding than applications written using the Windows API or Microsoft Foundation Classes (MFC). Additionally, with Windows Forms, a developer uses the same API regardless of which programming language they choose. Windows Forms offers full support for quickly and easily connecting to XML Web Services, as well as full support for the ADO.NET data model. Windows Forms and ASP.NET Web pages are developed in a similar fashion; however, ASP.NET is a browser-based solution that also requires a server, while Windows Forms can run independently on a client.

2.3.3 Common Language Specification

The Common Language Specification (CLS) is a set of basic language features needed by many applications. The CLS helps enhance and ensure language interoperability by defining a set of features that developers can rely on to be available in a wide variety of languages. Each language that is supported by the .NET Framework is compiled to the Microsoft Intermediate Language (MSIL), which is then converted to native code and executed on the CLR. This is illustrated in the following figure:

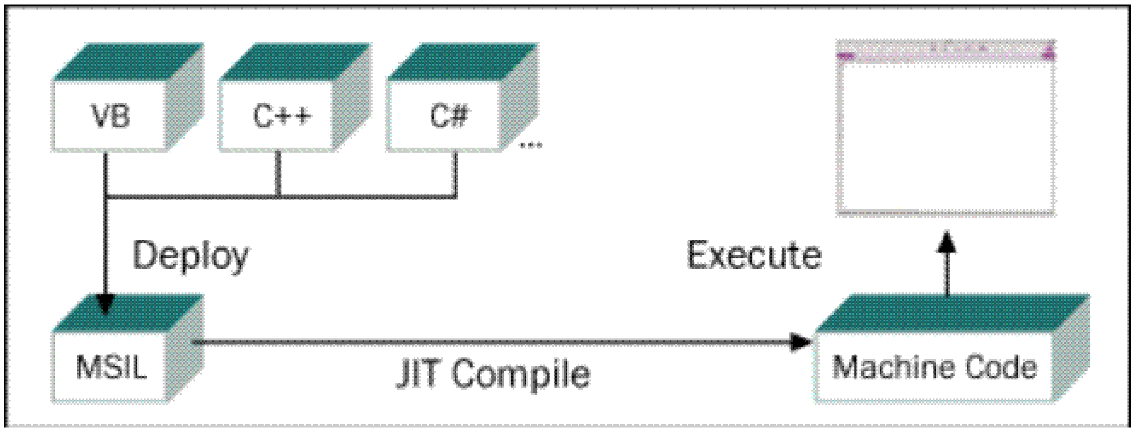

Figure 15

Source: Microsoft

With this approach, tools such as debuggers and profilers, are required to understand only one environment — the Microsoft intermediate language (MSIL) and metadata for the common language runtime — and they can support any programming language that targets the runtime.

2.3.4 Web Services Enhancements (WSE)

Web Services Enhancements for Microsoft .NET (WSE) is an add-on to Microsoft Visual Studio .NET and the Microsoft .NET Framework that provides developers with advanced Web Services capabilities. WSE 1.0 included support for Web Services protocols such as WS-Security, WS-Routing, Direction Internet Message Encapsulation (DIME), and WS-Attachments, as well as for security features such as username tokens and X.509 security tokens. WSE 2.0 introduces additional features including support for the WS-Addressing protocol, which defines transport-neutral mechanisms for identifying Web Service endpoints and securing end-to-end endpoint identification in messages, and support for Kerberos security tokens.

2.3.5 Servers and Web Services

This section provides a brief description of Web Services support in each of the following 3 Microsoft server products: Windows Server 2003, SQL Server, and BizTalk Server.

2.3.5.1 Windows Server 2003

Windows Server 2003 is first version of the Windows operating system that includes the .NET Framework with its installation. In addition to architectural changes in Internet Information Server (IIS), Windows Server 2003 contains a UDDI Services feature that enables companies to run their own UDDI directory for intranet or extranet use.

2.3.5.2 SQL Server 2003

SQL Server 2000 provides various options for reading XML data from SQL Server. Microsoft has also released SQLXML 3.0, a service pack for SQL Server that provides additional features to those already present in SQL Server 2000. SQLXML 3.0 introduces SOAP support that exposes SQL Server 2000 as a Web Service that offers SQL Server functionality to the client. SOAP HTTP requests can be sent to the server that is running SQLXML to execute stored procedures, user-defined functions (UDFs), and templates. For example, Visual Studio .NET can be used to access a WSDL service definition that is provided by SQLXML to automatically generate all of the classes that are necessary to access SQL Server as a Web Service, thus making it easy to call stored procedures and templates.

2.3.5.3 BizTalk Server 2002

Microsoft BizTalk Server 2002 is a Microsoft product for business process automation and application integration both within and between businesses. BizTalk Server 2002 provides a Web Services adapter that enables any server application to be accessible by using Web Services. BizTalk Adapter for Web Services is designed to work closely with the comprehensive data reformatting and application integration capabilities of Biz-Talk Server, and to provide a Web Service interface to almost any server application from any vendor. Additionally, with BizTalk Server 2002 Toolkit for Microsoft .NET, developers can build complete BizTalk projects using Visual Studio .NET and orchestrate ASP.NET XML Web Services.

2.3.6 Fundamental Differences between .NET and J2EE

The following are some fundamental differences between the .NET and J2EE frame-works:

- **Foundation:** .NET was built on the foundation of XML and Web Services, while J2EE (which originated before XML 1.0 was a W3C Recommendation and be-fore Web Services came into being) was not built with XML and Web Services at its core but rather has been extended over time to include support for XML and Web Services.
- **Language and Platform:** J2EE is Java-centric and platform neutral, while .NET is Windows-centric and language-neutral. This means developers are restricted to Java language in the J2EE and Windows in the .NET framework.
- **Specifications vs. Products:** J2EE is a series of specifications implemented by multiple vendors, while .NET is a set of Microsoft software technologies built on the foundation of XML and Web Services and offered by Microsoft products.
- **Platform Maturity:** J2EE can be considered a more mature platform, since it has existed longer than .NET.
- **Developer tools**: Multiple vendors offer J2EE developer tools, while Microsoft's Visual Studio .NET is the sole Integrated Development Environment (IDE) available for .NET.

2.4 References

GotDotNet: The Microsoft .NET Framework Community:
http://www.webservices.org/index.php/article/articleview/500/

MSDN:
http://msdn.microsoft.com/

Microsoft:
http://www.microsoft.com

3 Web Services Development Tools

This section describes Web Services development tools from various vendors. Web Services development tools enables software developers to execute tasks such as WSDL document creation, SOAP interface generation, and invocation and testing of Web Services. The tools listed in this section were chosen at random, and we do not endorse any particular vendor listed in this section.

3.1 IBM WebSphere SDK for Web Services (WSDK)

IBM WebSphere Software Developer Kit for Web Services V5.1 (WSDK V5.1) is an integrated kit for creating, discovering, invoking, and testing Web Services. WSDK V5.1 is designed to address the needs of experienced Java programmers who want to quickly learn how Web Services can be created using existing Java components and achieve seamless integration with disparate systems.

WSDK V5.1 can be used with the Eclipse IDE, which is an open-source, general-purpose, extensible IDE. WSDK V5.1 adds to the standard Eclipse package with tools relating to Web Services, making it more straightforward to build Web Services. WSDK V5.1 also enables developers to create and test Web Services that conform to the Final Approved WS-I Basic Profile 1.0. Web Services built with WSDK V5.1 can be extended and integrated using IBM WebSphere Studio V5.1, and deployed on IBM WebSphere Application Server V5.0.2.

WSDK V5.1 contains the following components:

- An embedded version of IBM WebSphere Application Server - Express V5.0.2, with additional support for Object Request Broker (ORB) and EJBs
- Support for SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, EJB 2.0, Enterprise Web Services 1.0 (JSR 109), WSDL4J, UDDI4J, and WS-Security
- IBM WebSphere UDDI v2.0 registry
- An entry-level database providing a JDBC implementation
- IBM SDK for Java 2 Standard Edition (J2SE) Technology, version 1.3.1
- Eclipse plug-ins to expose JavaBeans and stateless session EJBs as Web Services, to enable browsing for Web Services in UDDI registries, to create Web Services from WSDL definitions, and to publish and unpublish Web Services to a UDDI registry
- Command line tools to expose JavaBeans and stateless session EJBs as Web Services, to create Web Services from WSDL definitions, and to publish and unpublish Web Services to a UDDI registry
- Comprehensive documentation including Web Services concepts, developer tasks, and reference materials

3.2 Systinet WASP Developer

Systinet WASP (Web Applications and Services Platform) is a platform-independent, standards-compliant set of infrastructure products for building Java and C/C++ Web Services. Systinet WASP Developer seamlessly extends the industry's most popular Java IDEs (Borland JBuilder, Eclipse/IBM WebSphere Studio, and Sun ONE Studio/NetBeans) to support Web Services creation, debugging and deployment.

WASP Developer automates the generation of WSDL descriptions and SOAP interfaces, includes integrated deployment, debugging and monitoring tools, and provides support for UDDI query and publication. WASP Developer is a companion product for WASP Server for Java, which is a Web Services runtime environment for creating, deploying and managing Web Services in Java and J2EE applications.

The following are the main features of Systinet WASP Developer:

- Full support for Web Services standards such as SOAP 1.1, SOAP 1.2, WSDL 1.1 and WS-S
- Automatic Java > WSDL / WSDL > Java conversion
- Web Services Packaging and Deployment
- Client and Server-Side Debugging
- Seamless Deployment/Undeployment to WASP Server
- UDDI Wizards for Query & Publication

Systinet also has a "WASP Server for C++" product that is interoperable with both Java-based Web Services frameworks and Microsoft .NET. The following are the main features of Systinet WASP Server for C++:

- Full compliance with SOAP 1.1, SOAP 1.2, WSDL 1.1, SOAP with Attachments over MIME/DIME, and other leading standards
- Interoperability with Microsoft .NET, Apache SOAP/AXIS, and many others
- Portability across multiple platforms, including Linux, Solaris, HP-UX, Windows 32- bit systems, Windows CE/PocketPC, and many others
- Comprehensive security support for all industry-standard frameworks
- Ability to embed in existing C++ applications
- Full asynchronous Web Services support
- Tools for automation of Web Service creation and deployment

3.3 Microsoft SOAP Toolkit 3.0

The Microsoft SOAP Toolkit 3.0 consists of the following components:

- A client-side component that allows an application to invoke XML Web service operations described by a Web Services Description Language (WSDL) file. This

WSDL file describes the service(s) and operations of the service offered by the server.

- A server-side component that maps invoked XML Web service operations to COM object method calls as described by the WSDL and Web Services Meta Language (WSML) files. The WSML file is necessary for the implementation of the Microsoft SOAP Toolkit.
- The components needed to construct, transmit, read, and process SOAP messages. These processes are collectively referred to as *marshalling* and *unmarshalling*.

A WSML file provides information that maps the operations of a service (as described in the WSDL file) to specific methods in the COM object. The WSML file determines which COM object to load in order to service the request for each operation. In addition, the SOAP Toolkit provides a WSDL/WSML Generator tool that generates the WSDL and WSML files for you, relieving you of the tedious process of manually creating such files.

3.4 Java Web Services Developer Pack (Java WSDP) 1.3

The Java Web Services Developer Pack (Java WSDP) is a free integrated toolkit that enables Java developers to build and test XML applications, Web Services, and Web applications with the latest Web Services technologies and standards implementations. Technologies in Java WSDP include:

- The Java APIs for XML
- Java Architecture for XML Binding (JAXB)
- JavaServer Faces (simplifies building user interfaces for JavaServer applications)
- Web Services Interoperability (WS-I) Sample Application
- XML and Web Services Security
- JavaServer Pages Standard Tag Library (JSTL)
- Java WSDP Registry Server
- Ant Build Tool (Java-based built tool from Apache)
- Apache Tomcat (servlet container)

3.5 References

IBM WebSphere SDK for Web Services (WSDK) Version 5.1:
http://www-106.ibm.com/developerworks/webservices/wsdk/

Systinet WASP Developer:
http://www.systinet.com/products/wasp_developer/overview

Microsoft:

http://www.microsoft.com
Java 2 Platform, Enterprise Edition (J2EE):
http://java.sun.com/j2ee/

4 DISA SOAP Profile

In this section, we identify a SOAP profile that DISA could be using that can be practically implemented today, but should be continually aligned with future standards as they evolve.

We recommend that DISA implement the Web Services Interoperability Organization (WS-I) Basic Profile as its basis, and include the Basic Security Profile for security needs. Our primary reason for recommending this approach is the widespread backing for WS-I, with major vendors such as IBM, Microsoft, and Sun, which helps ensure a wide range of support in vendor products. This broad backing can also help ensure that the direction of these profiles will not radically change at any time.

Each of these profiles is described briefly below.

4.1 WS-I Basic Profile 1.0

The WS-Basic Profile 1.0 (hereafter referred to as “the Profile”) consists of a non-proprietary set of Web Services specifications, along with clarifications and amendments to those specifications that promote interoperability. While it is impossible to completely guarantee the interoperability of a particular service, the Profile does address the most common problems that implementation experience has revealed to date. The Profile's scope is initially bounded by the specifications referenced by it; “extensibility points” which are mechanisms or parameters that are identified but whose specification of usage is out of scope of the Profile then further refine it. Because the use of extensibility points may impair interoperability, their use should be negotiated or documented in some fashion by the parties to a Web service

The following topics addressed by the Profile are most pertinent to DISA:

- **Profile Conformance:** Conformance requirements at various levels to include:
    - Artifacts
    - Services, Consumers, and Registries
    - Messages
    - Registry Data
- **Messaging:** References the structure, content and handling of messages to include:
    - XML Representations of SOAP Messages
    - SOAP Processing Model
    - Use of SOAP in HTTP
- **Service Description:** Use of WSDL to enable the description of services as sets of endpoints operating on messages, to include:
    - Document Structure
    - Data Types
    - Bindings

- Usage of WSDL Elements

4.2 WS-I Basic Security Profile

The WS-Basic Security Profile 1.0 is a work in progress by the WS-I Basic Security Profile Working Group (BSPWG). Although the first version of the Basic Security Profile has not yet been released, we expect that it will incorporate the use of OASIS WS-Security, which we consider to be completely in line with the recommendations that we have made for DISA elsewhere in this paper.

4.3 References

Web Services Interoperability Organization (WS-I):
http://www.ws-i.org

5 Miscellaneous Web Services Products

This section describes multiple miscellaneous Web Services products, for the purpose of demonstrating the breadth of functionality that is currently covered in Web Services products. We have also selected one vendor for each product. The products listed in this section were chosen at random, and we do not endorse any particular vendor listed in this section.

The following types of Web Services products are covered:

- XML Authoring Tools (with Web Services features)
- Web Services-Based Business Integration Products
- Web Services Monitoring/Management Products
- Web Services Acceleration Products

5.1 XML Authoring Tools (with Web Services features)

XML authoring tools enable a user to develop XML artifacts such as XML Schemas, DTDs, and sample XML documents. While this "base" functionality is shared by all XML authoring tools, some contain more advanced features such as XML-to-database mapping, XML document comparison, and Web Services support. In this section, we describe an XML authoring tool that contains Web Services support.

5.1.1 Altova XML Spy 2004

Altova XML Spy 2004 is an XML Development Environment for building software applications based on XML technologies. Additionally, XML Spy 2004 includes full SOAP capabilities that include interpretation of WSDL (Web Services Description Language) documents, creation of SOAP requests, submitting them to a Web Service and viewing the SOAP Response, as well as a SOAP Debugger. XML Spy 2004 supports both Microsoft .NET and J2EE, and is optionally available with complete integration with Visual Studio .NET.

Below we describe several features of this product.

5.1.1.1 SOAP Debugger

The XML Spy 2004 SOAP Debugger acts as a Web Services proxy between a Web Services client and server, enabling a user to inspect WSDL files, single-step through Web Services transactions, inspect request and response XML documents, set breakpoints on SOAP functions, and define conditional breakpoints that trigger if a certain request or response contains selected data.

5.1.1.2 SOAP and WSDL Client Interface

This feature enables a user to open any existing WSDL document and immediately learn about the functions (i.e. SOAP operations) the corresponding Web Service provides (for example, "getLocation", "getTemperature", etc.). A user may then select these functions to automatically create a SOAP message, fill in the data, and send it to a Web Service. The resulting SOAP response message is then received and displayed.

5.1.1.3 WSDL Editor and Documentation Generator

XML Spy 2004 includes full WSDL editing which allows for easy visualization, validation, and documentation generation of WSDL files. The WSDL editor enables a user to edit, visualize and validate any WSDL file. The WSDL Documentation Generator makes it easier for a Web Service developer to document and publish a Web Service's interface to business partners, other developers, or to the public. A WSDL file can be easily annotated, then published into a Microsoft Word or HTML output file.

5.2 Web Services-Based Business Integration Products

Business integration products enable businesses to exchange data across their organizational boundaries, and to potentially integrate processes as well. Some of these products also contain XML authoring tools, thereby overlapping with the XML authoring tools described earlier. In this section, we describe a Web Services-based business integration product.

5.2.1 Cape Clear

Cape Clear Software's single focus is radically simplifying the integration of IT systems. Cape Clear Software's products have been built from the ground up using Web Services, and the company's product strategy is built around the growing importance of Business Services. Cape Clear defines Business Services as “business class Web Services that provide a high level, business oriented interface to a company's internal business processes and systems.”. Business Services are built using the Cape Clear Business Integration Suite.

Below we describe several features of Cape Clear 4.0.

5.2.1.1 Cape Clear Studio Toolkit

The Cape Clear Studio Toolkit simplifies and accelerates the process of integrating systems by enabling integrators to design Web Services, to create Web Services from existing components, and to combine Web Services into integrated business applications and to test, document and secure applications. The entry point to the Cape Clear Studio Toolkit is the Developer Center, which provides a task-oriented environment in which each step in the life-cycle is recorded as a task (using Apache Ant) to facilitate rapid service development and round-tripping.

In addition to the Developer Center, Cape Clear offers a variety of graphical tools:

- **WSDL Editor:** Provides wizards and validation tools to remove the complexity of creating and maintaining Web Service descriptions and performing web services design.
- **WSDL Generator:** Automatically creates a Web Service from an existing components such as EJBs, Java, or CORBA objects.

- **WSDL Assistant:** Automatically creates client proxies (Java or JSP) and server skeletons from a Web Service definition file in order to facilitate rapid consumption and prototyping of services.
- **Mapper:** Provides a simple interface for the creation of XSLT used to integrate Web Services with other XML and data interfaces.
- **UDDI Publisher and Browser:** Simplify the process of service discovery and publication.
- **WSDL Tester:** A general-purpose Web Service test tool that automatically creates a sample request for any Web Service, thus enabling users to create test data easily.
- **Server Deployment Tools:** Enable users to package and deploy services remotely and securely from within the development environment.

5.2.1.2 Cape Clear Server

Cape Clear Server is an application integration platform built on Web Services technologies. Features include:

- **Web Service Hosting:** Including sophisticated deployment and management features, as well as support for multiple service activation models. Services can be configured to per-request, per-session, per-user, or per-application models.
- **Routing:** Messages received by Cape Clear are routed based on rules configured statically on the server, dynamically based on message content or alternatively based on the message headers compliant with the WS-Routing specification.
- **Security:** Features include support for SSL and certificates, support for HTTP Basic authentication, a single sign-on and entitlements service, JAAS support for integration with 3rd party authentication and authorization systems and support for SAML assertions.
- **Message Transformations:** Support for advanced message transformation features enabling sophisticated syntactic and semantic mappings of incoming and outgoing data.
- **Middleware Adapters:** Provides the ability to map requests into invocations on various kinds of middleware including Java, J2EE and CORBA and MQ. Supports a wide range of middleware products and versions including BEA WebLogic, IBM WebSphere application server, IBM WebSphere MQ, Iona's Orbix and Borland's Visibroker.
- **UDDI:** Includes a UDDI 2.0 Registry used for publishing and discovery of Web Services.

5.2.1.3 Cape Clear Manager

Cape Clear Manager provides console-based security and management services to support mission-critical deployments including routing, diagnostics, authentication, and other crucial quality-of-service maintenance tools. All aspects of system configuration and administration are exposed as secure Web Services. Features include:

- A browser-based Manager console for all management functions.
- A "always on" model for maximum system availability.
- Administrator-configurable system logging.
- Dynamic endpoint assignment at deployment that allows WSDL to be updated automatically with remote host details prior to being published to UDDI.
- Support for secure multi-node remote deployment and configuration.

5.3 Web Services Monitoring/Management Products

Web Services monitoring/management products are primarily used for the monitoring and management of distributed Web Services. These products typically include additional capabilities such as policy enforcement and security. In this section, we describe a Web Services monitoring/management product.

5.3.1 AmberPoint Management Foundation

AmberPoint's Management Foundation product enables the management of complex business systems and their underlying Web Services. It allows operations personnel to monitor system-level activity such as performance, availability, and throughput, and gives IT users the ability to pause, restore or restart Web Services. AmberPoint Management Foundation also contains support for current and emerging security standards such as XML Encryption, XML Signatures, XACML and WS-Security.

AmberPoint uses a "non-invasive" approach that does not require any coding updates to a Web Service, and it also enables an organization to work with Web Services outside of their control. It provides consistent management capabilities for Web Services that run on disparate platforms such as J2EE and .NET. Management Foundation covers both business- and system-level management – i.e. it applies to both the business transactions (e.g. number of orders, type of requests) and system information (e.g. response time or throughput).

In addition to several other products, AmberPoint also has a distributed exception management solution called AmberPoint Exception Manager that detects, diagnoses and resolves operational and business exceptions in Web services systems ranging from simple data entry errors to complex business faults, such as orders lost in processing.

5.4 Web Services Acceleration Products

One of the tradeoffs regarding the many immense benefits of using XML and Web Services is that there is sometimes a price paid in terms of system performance due to the "verbosity" (use of tags and data) of XML. It should be noted that this is not always the case, as it depends on many factors to include the amount of data being transported, the length of XML tags, and bandwidth. Over the past several years there have been multiple "Web Services Acceleration" hardware-based products introduced that apply proprietary techniques in order to accelerate the processing of XML-based data and Web Services. Some of the products in this category also include security and management features, thereby overlapping with the Web Services monitoring/management products described earlier. In this section, we describe one such product.

5.4.1 DataPower XA35 XML Accelerator

The DataPower XA35 XML Accelerator is a network device that is capable of offloading overtaxed servers by processing XML and XSLT at "wirespeed". The XA35 may be installed in any number of different locations within an enterprise to offload Web and application servers from the arduous task of XML processing.

The following are some features of the XA35 XML Accelerator:

- **Advanced Application Support:** The XA35 supports XML/XSLT processing features such as XML Pipeline Processing™ to enable the most complex enterprise applications.
- **Application Transparency:** The XA35 provides "drop and insert" acceleration without any changes to application software
- **Web Services Security:** The XA35 can provide "fine-grained" XML access control for incoming network traffic and thwart XML-based denial-of-service attacks before they affect business servers. The XA35 also accelerates SSL encryption.
- **Advanced Management:** The XA35 provide real-time visibility into critical XML statistics such as throughput, transaction counts and other processing statistics.

The XA35 also integrates with several leading XML IDEs for the design, test, debugging, and deployment of Web Services.

5.5 References

Altova XML Spy:
http://www.xmlspy.com/

Cape Clear Software:
http://www.capeclear.com/

AmberPoint:

http://www.amberpoint.com
DataPower:
http://www.datapower.com

6 Conclusions

In this section, we have discussed a wide range of Web Services development products, environments, and tools, from application frameworks (such as J2EE and .NET), to development tools (such as SOAP toolkits), to products for Web Services monitoring/management and acceleration. We view Web Services not as a technology to ponder, but as a technology to utilize today. We believe that the Web Services standards landscape is still immature, but is rapidly maturing in critical foundational areas such as security and reliable messaging. We view Semantic Web technologies as extremely important yet still in their infancy. Despite the influx in both the Web Services standards and the Semantic Web technologies, we strongly recommend that DISA closely track the advancements being made in the wide range of Web Services development products, environments, and tools. While DISA may not require products in all of the discussed categories, we believe that there may be needs in many of the categories. Although we have not made specific product and tool recommendations in this section, we recommend that DISA undertake initiatives to evaluate products in the categories that have been discussed.